CODiNG TEST

2023/04/17 , 김혜진

이번 코딩테스트 ..

코테 본 기업

4월 15일(토) 10:00 NAVER

LG CNS 4월 16일(일) 15:00

코딩테스트 - NAVER

환경 세팅 & 입실 시작 **09:00**

주변 환경 360도 촬영 **09:45**

A4 용지 검사 **09:55**

시험 시작 **10:00**

시험 종료 **12:00**

2시간 동안 4문제

예상: 2솔

1문제는
테스트케이스 3개 중 2개 통과
근데 코드 제출하기를 안 눌렀다. ㅠㅠ

코딩테스트 - LG CNS

환경 세팅 & 입실 시작 **14:00**

주변 환경 360도 촬영 **감독 지시**

A4 용지 검사 **감독 지시**

시험 시작 **15:00**

시험 종료 **18:00**

3시간 동안 3문제

예상: 3솔

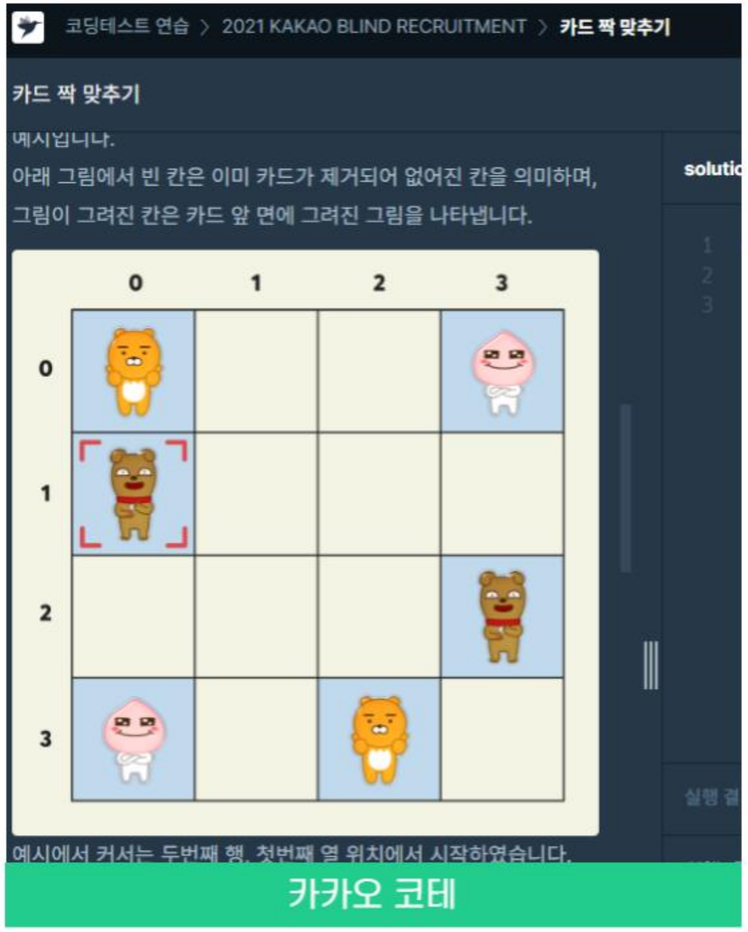
과연 ..

문제에 격자 그림이 나오면 극혐하던 나..

N사 마지막 문제에 격자
L사 마지막 문제에 격자

N사는 시간이 없어서
다른 문제에 집중하기로 함

L사는 시간이 생각보다 많이 남아서 도전!

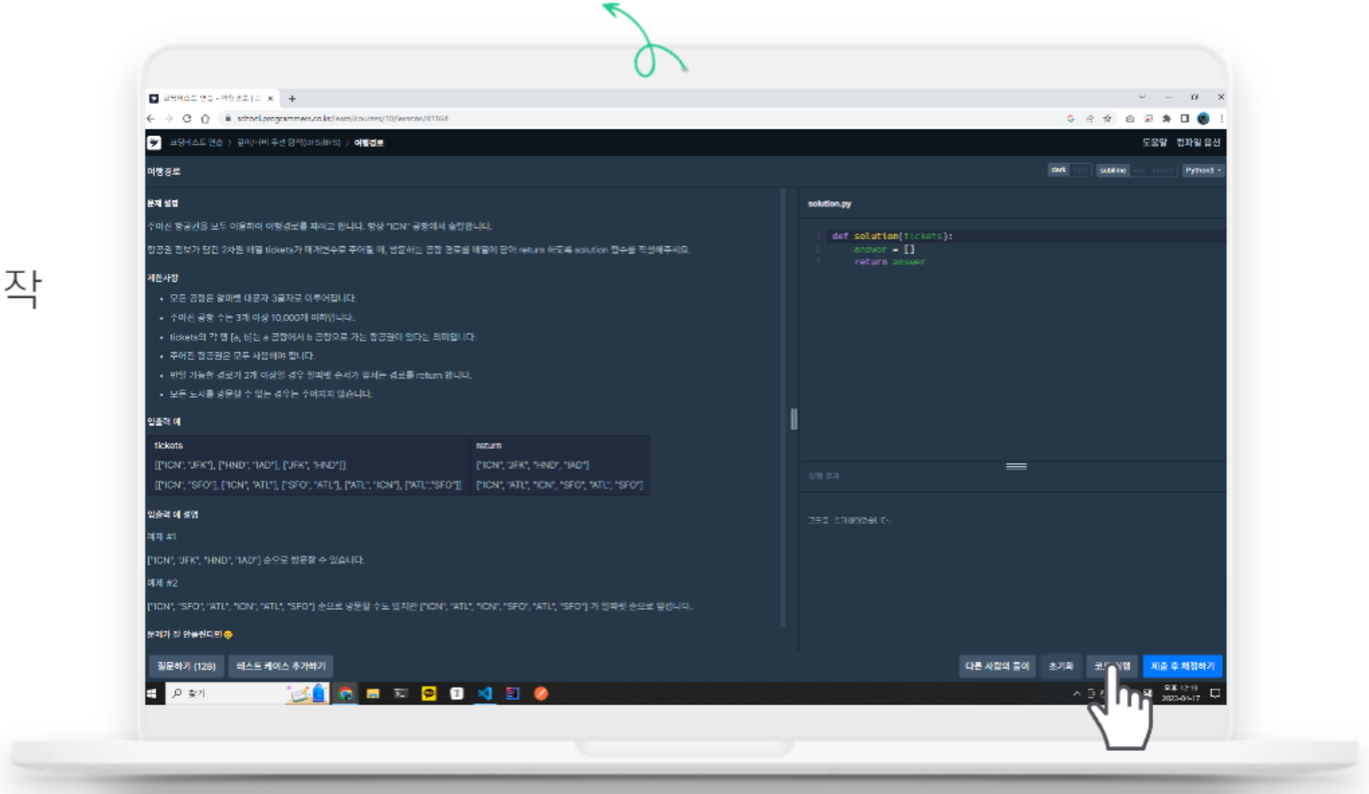


카카오 코테

커카오 인턴십 코테

코딩테스트 직전 다른 사람 풀이 찾아보고 공부한 문제

프로그래머스 Lv 3.여행 경로



코딩 테스트 직전 한 문제라도 풀어보자. 하고 풀기 시작

BFS/DFS 이용해서 풀다가 ..
도저히 반례를 찾지 못해서 다른 사람의 풀이를 봤다.

완벽하게 이해하지는 못했지만, 코테 때
'어 이거 방금 봤던 거랑 비슷하게 풀면 되겠다.' 하고
자신감을 얻음

프로그래머스 Lv3. 여행 경로

문제 설명

주어진 항공권을 모두 이용하여 여행경로를 짜려고 합니다. 항상 "ICN" 공항에서 출발합니다.

항공권 정보가 담긴 2차원 배열 tickets가 매개변수로 주어질 때, 방문하는 공항 경로를 배열에 담아 return 하도록 solution 함수를 작성해주세요.

제한사항

- 모든 공항은 알파벳 대문자 3글자로 이루어집니다.
- 주어진 공항 수는 3개 이상 10,000개 이하입니다.
- tickets의 각 행 [a, b]는 a 공항에서 b 공항으로 가는 항공권이 있다는 의미입니다.
- 주어진 항공권은 모두 사용해야 합니다.
- 만일 가능한 경로가 2개 이상일 경우 알파벳 순서가 앞서는 경로를 return 합니다.
- 모든 도시를 방문할 수 없는 경우는 주어지지 않습니다.

입출력 예

tickets	return
[["ICN", "JFK"], ["HND", "IAD"], ["JFK", "HND"]]	["ICN", "JFK", "HND", "IAD"]
[["ICN", "SFO"], ["ICN", "ATL"], ["SFO", "ATL"], ["ATL", "ICN"], ["ATL","SFO"]]	["ICN", "ATL", "ICN", "SFO", "ATL", "SFO"]

프로그래머스 Lv3. 여행 경로

아이디어 1

1. ticket_dict = { 출발지: [도착지] } 저장
2. ticket_dict[출발지] 정렬
3. ticket_dict와 짝을 이루는 visited_dict = { 출발지: [0, 0, ..., 0] } 생성
4. "ICN"을 시작으로 DFS 탐색하며 visited에 방문 순서 저장
5. 방문 순서 출력

DFS

출발지를 받아 ticket_dict[출발지] 리스트의 도착지들을 순회하며
visited_dict[출발지][도착지]가 0이라면 visited에 넣고 방문 처리

```python
def solution(tickets):
    answer = []
    ticket_dict = {}
    for arr, dep in tickets:
        ticket_dict.setdefault(arr, [])
        ticket_dict[arr].append(dep)
    for arr in ticket_dict:
        ticket_dict[arr].sort()
    visited_dict = {k: [0]*len(ticket_dict[k]) for k in ticket_dict}
    visited = ["ICN"]
    def dfs(arr):
        nonlocal visited, visited_dict
        if arr not in ticket_dict: return
        for idx, dep in enumerate(ticket_dict[arr]):
            if not visited_dict[arr][idx]:
                visited_dict[arr][idx] = 1
                visited.append(dep)
                dfs(dep)
    dfs(visited[0])
    return visited
```

프로그래머스 Lv3. 여행 경로

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉	실패 (0.03ms, 10.1MB)
테스트 2 〉	실패 (0.01ms, 10.3MB)
테스트 3 〉	통과 (0.01ms, 10.1MB)
테스트 4 〉	통과 (0.01ms, 10.3MB)

채점 결과

정확성: 50.0

합계: 50.0 / 100.0

프로그래머스 Lv3. 여행 경로

아이디어 2

1. tickets와 짝을 이루는 visited 이용
2. "ICN"을 시작으로 DFS 탐색하며 answer에 방문 순서 저장

DFS

(tickets[i][0] == 출발지)인 도착지들 중,
이름이 가장 빠른 것을 찾아 출발지로 지정하여 반복

```python
def solution(tickets):
    visited = [0 for _ in tickets]
    answer = ["ICN"]
    while len(answer) != len(tickets)+1:
        arr = answer[-1]
        dep_idx, dep = -1, "ZZZ"
        for idx, ticket in enumerate(tickets):
            if ticket[0] == arr and not visited[idx]:
                if dep > ticket[1]:
                    dep = ticket[1]
                    dep_idx = idx
        visited[dep_idx] = 1
        answer.append(dep)
    return answer
```

프로그래머스 Lv3. 여행 경로

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉	실패 (0.04ms, 10.2MB)
테스트 2 〉	실패 (0.01ms, 10.3MB)
테스트 3 〉	통과 (0.01ms, 10.1MB)
테스트 4 〉	통과 (0.01ms, 10.2MB)

채점 결과

정확성: 50.0

합계: 50.0 / 100.0

프로그래머스 Lv3. 여행 경로

반례

입력값 〉	[["ICN", "JFK"], ["ICN", "AAD"], ["JFK", "ICN"]]
기댓값 〉	["ICN", "JFK", "ICN", "AAD"]
실행 결과 〉	실행한 결괏값 ["ICN","AAD","JFK","ICN"]이 기댓값 ["ICN","JFK","ICN","AAD"]과 다릅니다.

이름이 빠른 걸 기준으로 먼저 탐색했는데,
AAD까지 갔다가 경로가 없으니까 다시 ICN으로 돌아가서 JFK로 갔다.
하지만 AAD에서 JFK까지 가는 경로는 없다.

즉 더이상 경로가 없는 애는 맨 마지막으로 빼야 함.

프로그래머스 Lv3. 여행 경로

아이디어 3

DFS로 탐색하다가 더이상 경로가 없는 공항은
remain에 저장 후 마지막에 더한다.

DFS

반복 별로 현재 출발지에서 갈 수 있는
arrival_dict={도착지: tickets 기준 index)를 구한다.
arrival_dict가 비었으면 remain에 저장하고,
아니라면 arrival_dict를 key값 기준으로 정렬하여
첫 번째 값을 answer에 넣고, tickets에서 제거한다.

```python
def solution(tickets):
    answer = ["ICN"]
    remain = [] # 더이상 이동하지 못하는 공항 저장

    while tickets:
        departure = answer[-1]

        arrival_dict = {} # departure에서 갈 수 있는 (공항: 인덱스) 쌍 저장
        arrival_dict = { ticket[1]: idx
                        for idx, ticket in enumerate(tickets)
                        if ticket[0] == departure }

        if not arrival_dict:    # departure에서 갈 수 있는 공항이 없는 경우
            remain.append(answer.pop()) # departure를 빼서 remain에 저장
        else:                   # departure에서 갈 수 있는 공항이 있는 경우
            arrival_list = sorted(arrival_dict.items(), key=lambda x: x[0]) # 공항명을 기준으로 정렬
            arrival = arrival_list[0] # 목적지 (이름, 인덱스)
            answer.append(arrival[0])
            tickets.pop(arrival[1])

    if remain: answer += remain
    return answer
```

프로그래머스 Lv3. 여행 경로

입력값 〉	[["ICN", "JFK"], ["ICN", "AAD"], ["JFK", "ICN"]]
기댓값 〉	["ICN", "JFK", "ICN", "AAD"]
실행 결과 〉	테스트를 통과하였습니다.

실행 결과

채점을 시작합니다.

정확성 테스트

테스트 1 〉	실패 (0.08ms, 10.1MB)
테스트 2 〉	통과 (0.02ms, 10.2MB)
테스트 3 〉	통과 (0.01ms, 10.2MB)
테스트 4 〉	통과 (0.02ms, 10.1MB)

채점 결과

정확성: 75.0

합계: 75.0 / 100.0

프로그래머스 Lv3. 여행 경로

이제 슬 시험 시간도 다가오고 ..

다른 사람 풀이를 찾아 이해하고 시험쳤다.

근데 (방금) 내 코드에서 문제를 찾아 고쳤다!!

테스트 3	
입력값 〉	[["ICN", "A"], ["A", "B"], ["A", "C"], ["B", "A"], ["C", "A"]]
기댓값 〉	["ICN", "A", "B", "A", "C", "A"]
실행 결과 〉	테스트를 통과하였습니다.

테스트 4	
입력값 〉	[["ICN", "BOO"], ["ICN", "COO"], ["COO", "DOO"], ["DOO", "COO"], ["BOO", "DOO"], ["DOO", "BOO"], ["BOO", "ICN"], ["COO", "BOO"]]
기댓값 〉	["ICN", "BOO", "DOO", "BOO", "ICN", "COO", "DOO", "COO", "BOO"]
실행 결과 〉	테스트를 통과하였습니다.

입력값 〉	[["ICN", "A"], ["A", "B"], ["A", "C"], ["C", "A"], ["B", "D"]]
기댓값 〉	["ICN", "A", "C", "A", "B", "D"]
실행 결과 〉	실행한 결괏값 ["ICN","A","C","A","D","B"]이 기댓값 ["ICN","A","C","A","B","D"]과 다릅니다.

프로그래머스 Lv3. 여행 경로

아이디어 3

DFS로 탐색하다가 더이상 경로가 없는 공항은
remain에 저장 후 마지막에 더한다.

DFS

...

```python
def solution(tickets):
    answer = ["ICN"]
    remain = [] # 더이상 이동하지 못하는 공항 저장

    while tickets:
        departure = answer[-1]

        arrival_dict = {} # departure에서 갈 수 있는 (공항: 인덱스) 쌍 저장
        arrival_dict = { ticket[1]: idx
                        for idx, ticket in enumerate(tickets)
                        if ticket[0] == departure }

        if not arrival_dict:    # departure에서 갈 수 있는 공항이 없는 경우
            remain.append(answer.pop()) # departure을 빼서 remain에 저장
        else:                   # departure에서 갈 수 있는 공항이 있는 경우
            arrival_list = sorted(arrival_dict.items(), key=lambda x: x[0]) # 공항명을 기준으로 정렬
            arrival = arrival_list[0] # 목적지 (이름, 인덱스)
            answer.append(arrival[0])
            tickets.pop(arrival[1])

    if remain: answer += remain
    return answer
```

프로그래머스 Lv3. 여행 경로

아이디어 4

DFS로 탐색하다가 더이상 경로가 없는 공항은 remain에 저장 후 마지막에 **뒤집어서** 더한다.

DFS

...

```python
def solution(tickets):
    answer = ["ICN"]
    remain = [] # 더이상 이동하지 못하는 공항 저장

    while tickets:
        departure = answer[-1]

        arrival_dict = {} # departure에서 갈 수 있는 (공항: 인덱스) 쌍 저장
        arrival_dict = { ticket[1]: idx
                         for idx, ticket in enumerate(tickets)
                         if ticket[0] == departure }

        if not arrival_dict:    # departure에서 갈 수 있는 공항이 없는 경우
            remain.append(answer.pop()) # departure를 빼서 remain에 저장
        else:                   # departure에서 갈 수 있는 공항이 있는 경우
            arrival_list = sorted(arrival_dict.items(), key=lambda x: x[0]) # 공항명을 기준으로 정렬
            arrival = arrival_list[0] # 목적지 (이름, 인덱스)
            answer.append(arrival[0])
            tickets.pop(arrival[1])

    if remain: answer += remain[::-1]
    return answer
```

프로그래머스 Lv3. 여행 경로

정확성 테스트

테스트 1 〉 통과 (0.03ms, 10.2MB)

테스트 2 〉 통과 (0.03ms, 10.3MB)

테스트 3 〉 통과 (0.01ms, 9.97MB)

테스트 4 〉 통과 (0.01ms, 10.1MB)

채점 결과

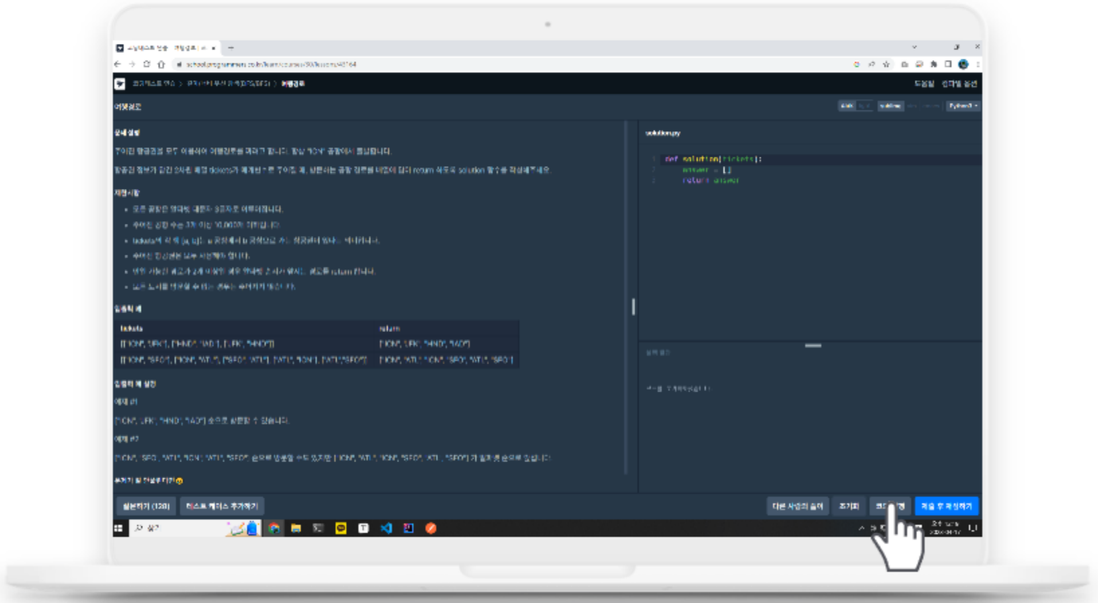
정확성: 100.0

합계: 100.0 / 100.0

후기

실전 코딩테스트에서는 테스트 케이스만 잘 통과하면 제출하겠지만..
애초에 테스트 케이스에 맞춘 풀이가 아니라,
알고리즘 자체를 생각하고 푸는 연습을 하여
반례를 찾아낼 줄도 알아야 한다고 생각한다.

그리고 그림에 압도당하지 말 것!!
그저 이해를 돕는 거라고 생각해야 한다.



THANK YOU